

A COMPARISON OF ULTRACOMPUTER ARCHITECTURE AND LATTICE
ARCHITECTURE FOR PROBLEMS
ON LATTICES

Many numerical methods make use of a computational mesh in the form of a square or cubic lattice in two or three space dimensions. The equations are generally difference equations which couple each point of the lattice to its neighbors. This suggests a natural architecture for a parallel machine in which the processors are arranged in a corresponding lattice. In 3 dimensions, with $N = n^3$ processors, the processor (i,j,k) is connected to $(i\pm 1,j,k)$, $(i,j\pm 1,k)$, and $(i,j,k\pm 1)$. We shall call this a lattice computer.

In an ultracomputer, by contrast, each processor is given an index $\ell = 0, 1, \dots, N-1$, where $N = 2^P$. Processor ℓ is connected to $(\ell+1) \bmod N$, $(\ell-1) \bmod N$, $\sigma(\ell)$, and $\sigma^{-1}(\ell)$, where $\sigma(\ell)$ is the shuffle permutation which is defined as a left cyclic shift of the binary digits of ℓ .

These two architectures have in common the fact each processor is connected to a fixed (independent of N) number of others, so it is fair to compare them.

We begin by describing the kind of numerical method for which a lattice machine is well-suited. Suppose, for ex-

ample, we use an explicit numerical method to solve the time-dependent equations of gas dynamics. At each time step, we update the density and velocity according to their values at neighboring mesh points. On a lattice machine this requires a fixed number of machine cycles per time step, which is certainly optimal.

Next, we consider how such a method would be implemented on an ultracomputer. Suppose we associate the mesh point (i,j,k) with processor $\ell = i + nj + n^2 k$ where $i,j,k = 0,1,\dots,n-1$ and $n = 2^q$. Then the number of processors is $N = n^3 = 2^{3q}$. To execute the numerical method, processor ℓ needs access to the information stored in $\ell \pm 1$, $\ell \pm n$, and $\ell \pm n^2$. Only the first of these is directly available through the ultracomputer connections. Fortunately, the others are available at the expense of a certain number of shuffle operations.

To see this, let $b(i)$ be the binary representation of i , and similarly for j , k , ℓ . We have

$$b(\ell) = b(k)b(j)b(i)$$

where the right-hand side just means the three binary numbers written in sequence as a single number. Since each shuffle operation is defined by a left cyclic shift of the binary digits, we have

$$b(\sigma^q(l)) = b(j)b(i)b(k)$$

In other words, q shuffles accomplish, in effect, a cyclic permutation of the coordinate axes. This means that the neighbors in all three space directions are available at the expense of $3q = \log_2 N$ shuffle operations. (The third permutation is used to bring the axes back to their original configuration.) Thus an ultracomputer can simulate a lattice machine at the expense of only a $\log N$ factor in speed.

It should be noticed at this point that the same ultracomputer can be used for two-dimensional or three-dimensional problems without rewiring. The only requirement is that $\log_2 N$ be a multiple of 2 as well as 3. Then, for a two-dimensional problem we can set $l = i + n'j$ where $n' = 2^{q'}$ and $N = (n')^2 = 2^{2q'}$.

The real strength of the ultracomputer in numerical analysis shows up, however, when the physical problem has an equilibrium aspect so that a system of simultaneous equations (linear or nonlinear) has to be solved either once (in a static problem) or at each time step (in a time-dependent problem.) Such problems arise in the theory of elasticity, in incompressible fluid dynamics, in quantum chemistry, and in numerical weather prediction. More generally, simultaneous systems of equations arise in

every static problem and whenever an implicit numerical method is used on a time-dependent problem.

The characteristic feature of simultaneous systems of equations is that the solution at one point depends on the data everywhere. This is true even when the problem is formulated in such a way that points are coupled only to neighbors; the inverse of a sparse matrix is usually dense, for example. No matter how such a problem is solved, each item of data must move in such a way that it can influence all of the outputs. In a lattice machine with n points in each direction it is clear that this takes at least $O(n)$ machine cycles. An ultracomputer, on the other hand, can disseminate information throughout the machine in $\log N$ cycles. This gives the ultracomputer a decisive advantage for large N .

We conclude that for most numerical problems on lattices, an ultracomputer will be faster than a lattice machine, despite the fact that the lattice machine seems to have precisely the right architecture for the problem. There is a special class of problems where the lattice machine is faster, but only by a factor of $\log N$. In either case, the ultracomputer will be a more flexible machine, adaptable to a variety of problems without rewiring.

1. 2. 3.

4.

5.

6.

7.

8.

9.

10.

11.

12.

13.

14.

15.

16.

17.

18.

